OLE and the SAS® System for Windows Release 6.12

Jennifer Clegg and Carol Rigsbee, SAS Institute Inc., Cary, NC

ABSTRACT
This paper describes the OLE support within the SAS System for Windows Release 6.12. This support includes OLE container, OLE automation controller, and OLE automation server functionality. OLE container and OLE controller support is available in SAS/AF® FRAME entries and SAS/EIS® applications. OLE automation server support is available with base SAS software.

INTRODUCTION
This paper provides an overview of the OLE support in the SAS System. For more details on OLE in general, see the documentation for the Microsoft Windows operating environment. For further details on OLE support within the SAS System, refer to the online documentation for the SAS Companion.

OLE facilitates the exchange of information between applications. OLE containers include objects and controls from other applications. Through OLE automation, OLE controllers script objects in other applications. OLE servers are the applications that provide these objects.

OLE functionality within the SAS System falls into three categories: basic container support, OLE automation support, and OLE controls support. Each section of this paper discusses the particular OLE feature followed by specific information about its use within the SAS System.

BASIC CONTAINER SUPPORT

Overview of OLE
The SAS System’s basic container supports the creation of Linked and Embedded objects, drag and drop, and visual editing.

Objects can be Linked or Embedded depending on your needs. Use Linked objects when you want the data in the container to update dynamically when the source changes. Use Embedded objects if you want to edit the object within the container or if a link would be unavailable later.

Drag and drop provides an easy way to create an Embedded or Linked object within a container. Drag and drop supports keyboard modifiers to alter the behavior of a drop. By default, dragging an object moves the object. To copy an object, hold down the Ctrl key when you drop the object in the container. The cursor will change to an arrow with a box and a plus (+) sign. To create a link to the object data, hold down the Ctrl and Shift keys when you perform the drop. The cursor will become an arrow with a box and an equals (=) sign. If the target area is not a valid drop site or the operation is not supported, the cursor will change to the not (∅) sign.
Visual editing allows you to edit an object within the context of its container. The container takes on the user interface of the server. The menus, toolbars, and status line switch to the ones normally provided by the server. Below is an example of a Microsoft Word object being visually edited in the SAS container.

Not all servers support visual editing. Some servers may support open editing where the server launches as a separate application and all editing occurs in that application’s window. By design, Linked objects only support open editing. Embedded objects may support visual editing or open editing depending on the server.

Overview of OLE and the SAS System
There are multiple ways to create Embedded and Linked objects with the SAS System. First, you need to bring up the SAS BUILD:DISPLAY window so you can create a FRAME entry that will contain your OLE objects. To do this submit the following statement:

```
proc build c=sasuser.ole.demo.frame; run;
```

To create OLE objects you can select FILL or MAKE from the frame’s popup menu. To access the popup menu, position your mouse in the BUILD window and click the right mouse button. When using FILL, you must first drag out a region to designate the size of the object. When using MAKE, the server determines the default size of the object. For bitmaps and other objects that do not scale well, MAKE is preferable because the object retains its appearance.

The SAS System creates HSERVICE entries in SAS Catalogs to store the necessary information about Linked and Embedded objects. These entries are only portable to other Windows platforms.

Creating OLE Objects in FRAME
One of the ways you can create OLE objects in FRAME is to use the Insert Object dialog. To access this dialog, select MAKE or FILL from the frame’s popup menu. Scroll through the list of items and select OLE - Insert Object. The dialog is below.

```
Create New

Create from Document

Insert Control

Insert Object

OK

Cancel

Help

Display As Icon

Inserts a new linked Object into your document.
```

Notice that there are three radio buttons in this dialog. The first one is Create New. Selecting this button will display a list of available object types. This list will vary based on the OLE-capable applications on your machine. Selecting an object type from this list and pressing OK will result in the creation of an Embedded object. Since this is a new instance of an object, the server starts the object in edit mode, either visual editing or open editing, depending on what the server supports. Click outside the object, else-
where on the frame, to end visual editing. Exit the server to end open editing.

If you select the second radio button, Create from File, the dialog allows you to create an object based on the contents of a file. The dialog prompts you for the filename. The Insert Object dialog with this option selected is below.

If you do not know the filename, you can browse for the file to use to create this object. Using this method will create an Embedded object based on the contents of the file. To create a Linked object, click the Link check box to select it.

The OLE Controls section describes the third radio button, Insert Control.

Another way to create a Linked or Embedded object is to use the Paste Special dialog. This dialog is available from the MAKE or FILL menus as OLE - Paste Special.

The Paste Special dialog allows you to create an object based on data from the clipboard. You may create objects of the following types: Embedded, Linked, Metafile, Device Independent Bitmap, or Bitmap. Select the desired choice and click the Paste button to create any type of object except Linked. You must select the Paste Link radio button to create a Linked object. Metafile, Device Independent Bitmap, and Bitmap are all static data representations of an object. You may view these objects in a container but not edit them.

Another way to create an object is to read an existing object from a SAS catalog. To read an object from a catalog select OLE - Read Object from the MAKE or FILL menus. Enter the HSERVICE entry name of the object you want to read. Reading an object from a catalog is equivalent to copying an object. This method allows you to access objects in other frames. You must change the name of the HSERVICE entry in the Object Attributes dialog after you create the object if you want to save the object as a separate catalog entry.

You can also use drag and drop to create a new object based on an existing object or data in an application.

**OLE Verbs**

One important feature of OLE objects is that they expose verbs. Verbs are actions that can be performed on an object. Each OLE object has a default verb associated with it. Most objects expose “Edit” as the default verb. Some objects, such as media clip, expose “Play” as the default verb. The default verb is important because double-clicking an object in TESTAF executes this action. Most objects have more than one verb.

To access all the verbs for an object in BUILD, click on the object with the right
mouse button to access the frame’s menu. The name of the OLE object is the last item on the menu. This menu contains a list of selections available with this object. Below is a fragment of the frame’s menu showing the object’s menu.

The first set of selections (before the separator) is the list of verbs. The first selection is the default verb. You can select any of these verbs to execute that action. Additional selections available from this menu include the Links and Convert dialogs.

You can also view the list of verbs for an object by invoking the Object Attributes dialog and selecting the Associated Verbs item. Using SCL, you can programmatically execute verbs using the _EXECUTE_ method. An example is below:

```plaintext
call notify('oleobj', '_EXECUTE_', 'Edit');
```

The Convert dialog allows you to specify an alternate editor for an object when the original server is unavailable. If you convert an object, it permanently becomes an object of the new type. You can also use this dialog to activate an object as an object of another type. The storage for the object is unaltered and all objects of this type on your system will now use the new application for editing. Conversion and activation support depends on the servers installed on your system. One way to access the Convert dialog is to select an object and type DLGCONVERT on a command line. You also can use SCL to pass DLGCONVERT as a verb to the object. Additionally, in BUILD, you can access the Convert dialog from the object’s menu.

**OLE AUTOMATION**

Automation allows an application to script another application. An OLE Automation Server is an application that
exposes objects for scripting. These objects are OLE Automation Objects. Applications that access these objects are OLE Automation Controllers. Controllers provide a mechanism such as a script language to access automation objects and program automation servers.

You can control an OLE Automation Object through its methods and properties. Methods are functions exposed by an object. Properties are variables exposed by an object. The server’s documentation describes its objects and their methods and properties.

SAS OLE Automation Server
The SAS System’s automation server exposes one automation object to allow access to some features of the SAS System. The SAS System’s automation object supports the following methods and properties:

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command</td>
<td>executes the text as if it were entered on the command line of the SAS System session</td>
</tr>
<tr>
<td>Submit</td>
<td>submits text to the SAS automation server for execution by the program editor</td>
</tr>
<tr>
<td>Quit</td>
<td>ends the SAS automation session</td>
</tr>
<tr>
<td>QueryWindow</td>
<td>determines whether a specified window exists within the SAS automation session</td>
</tr>
<tr>
<td>Top</td>
<td>makes the SAS automation session the topmost application</td>
</tr>
</tbody>
</table>

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>controls the visibility of the automation session</td>
</tr>
<tr>
<td>Title</td>
<td>specifies the text for the main title bar</td>
</tr>
<tr>
<td>Busy</td>
<td>indicates whether the automation session is currently doing work and is unable to process additional Command or Submit requests immediately</td>
</tr>
<tr>
<td>Wait</td>
<td>determines when control will be returned to the caller</td>
</tr>
</tbody>
</table>

CommandWindow
specifies the title of the window, e.g. “Program Editor”, that will receive the command statements specified in the Command method

CommandWindowVisible
determines the visible status of the CommandWindow, if it is specified

RC
contains a return code that is set in the SAS System session using the “setrc” user written function

ResultString
contains some text that is set in the SAS System session using the “setrc” user written function

Parent
specifies the parent window of the SAS application session

X
x position of SAS session window

Y
y position of SAS session window

Width
width of SAS session window

Height
height of SAS session window

See Appendix A for a Visual Basic example that scripts the SAS System automation server.

The SAS System installation process updates the registration database with information necessary for the automation server. To start the server from an automation controller, you need its ProgID (programmatic identifier). The identifier for the server is “SAS.Application.612”. Automation server support is available with base SAS software and is supported only on the Windows 95 and Windows NT platforms.

It is possible to have multiple automation controllers accessing one instance of an automation server. In Visual Basic, for example, the mechanism for accessing an existing instance of an automation server is GetObject, instead of creating a new instance with CreateObject.
One caveat is that the SAS System automation session will not close down until all the automation controllers that are accessing it have issued Quit requests or have closed down. For this reason, if the SAS System automation session is visible, be careful before manually shutting it down by double clicking the system icon or selecting Exit from the File menu. There is no way for the SAS automation server to notify controllers that the SAS automation session is going away, and the automation controllers may continue attempting to access it.

**SAS OLE Automation Controller**

As an OLE Automation Controller, the SAS System must provide a way to access OLE Automation servers and their OLE Automation Objects. SAS/AF provides the following SCL methods for this access:

- **_NEW_** creates an OLE Automation object
- **_SET_PROPERTY_** sets the value of a property
- **_GET_PROPERTY_** gets the value of a property
- **_DO_** invokes method that does not return a value
- **_COMPUTE_** invokes method that does return a value
- **_GET_REFERENCE_ID_** returns SCL reference identifier for the object

See Appendix B for an SCL example that uses these methods. You can also automate Linked or Embedded objects that exist in your FRAME by using these same SCL methods.

Many applications support Visual Basic to automate OLE servers. For example, you can use the macro recorder in Excel to capture the Visual Basic commands necessary to perform some functions. This code quickly translates into SCL.

See Appendix C for an example of equivalent SCL and Visual Basic commands.

**OLE CONTROLS**

OLE controls, OCXs, are Embedded OLE objects with additional functionality. The main additions are support for ambient properties and events. Ambient properties allow communication between a container and an OCX for notification of property changes such as fonts and colors. For example, if the font in the container changes, the container will notify the OCX so it can change its font if applicable. Events are notifications generated by the OCX to the container. For example, a push button control might generate a click event.

Like Linked and Embedded objects, an HSERVICE entry within a SAS catalog stores the object information. When distributing a catalog that contains an OCX, you must copy the catalog as well as the OCX.

To use an OCX in a frame, you use the Insert Object dialog that was discussed earlier in the basic container section. From within this dialog select the Insert Control radio button. The dialog with this option selected is below.

![Insert Object Dialog](image-url)
The list of controls displayed will vary based on the controls installed on your machine. You can register new controls by selecting the Add Control button. Select a control from the list and click OK to create an instance of it in the frame. Alternatively, you can drag and drop a control in the frame from another application or paste one in from the clipboard using OLE - Paste Special.

OCXs provide a property sheet for access to some of the control’s properties. To access the property sheet in BUILD, select the Properties verb from the object’s menu. To access the property sheet using SCL, you send the Properties verb to the object using the _EXECUTE_ method.

To access all the properties and methods of a control, you use a subset of the SCL methods provided for automation. The SCL methods _GET_PROPERTY_ and _SETPROPERTY_ access the properties of a control. The _DO_ and _COMPUTE_ SCL methods invoke methods of a control. You must know the names of the properties and methods for the object as well as any arguments. The documentation for the OCX should include this information.

OCXs generate events that you can respond to in your SCL code. In the SAS System you can view and map these events using the Event Map dialog. This dialog is below.

To access this dialog invoke the Object Attributes dialog from the frame’s popup menu and select Event Map. You may now select an event and specify the name of the SCL, FRAME, or PROGRAM source entry and (if applicable) the SCL label where the event-handling code resides. You can edit the SCL for the event from this screen by pressing “Edit”. When you map an event, you must compile the SCL outside of the frame in which the control exists.

**CONCLUSIONS**

This paper summarizes the support for OLE within the SAS System for Windows Release 6.12. Both OLE container support and OLE automation server support facilitate the exchange of information between SAS and other applications. OLE container support allows the use of third party developed objects within SAS/AF and SAS/EIS applications. OLE automation server support allows other third party applications to access some features of the SAS System.
REFERENCES

SAS, SAS/AF, and SAS/EIS are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand or product names are registered trademarks or trademarks of their respective companies.

Jennifer Clegg
SAS Institute, Inc.
100 SAS Campus Drive
Cary, NC 27513
(919) 677-8000
e-mail: jbc@unx.sas.com

Carol Rigsbee
SAS Institute, Inc.
100 SAS Campus Drive
Cary, NC 27513
(919) 677-8000
e-mail: sasczw@unx.sas.com
APPENDIX A

This Visual Basic example creates an instance of the SAS automation server, submits code to the server, and closes down the server.

REM Create a button to invoke an automation server instance.
Private Sub StartBtn_Click()
    Set SASObj = CreateObject("SAS.Application.612")
    SASObj.Title = "Hat Graph"
    SASObj.Visible = TRUE
End Sub

REM Create a button to perform the calculations.
Private Sub ComputeHat_Click
    SASObj.Submit ("data hat;do x=-5 to 5 by .25;"
    SASObj.Submit ("do y=-5 to 5 by .25;z=sin(sqrt(x*x+y*y));"
    SASObj.Submit ("output; end; end;"
    SASObj.Submit ("proc g3d data=work.hat;plot y*x=z/ctop=red;"
    SASObj.Submit ("Title 'Cowboy Hat';run;quit;"
End Sub

REM Create a button to end the automation server instance.
REM Visual Basic requires that you set the object holder to Nothing to REM break the connection.
Private Sub EndSAS_Click()
    SASObj.quit
    Set SASObj = Nothing
End Sub

APPENDIX B

This SCL example invokes Microsoft Excel, opens a worksheet, selects a range of cells, and charts them.

length rangeid $80;

/* create excel automation object */
hostcl = loadclass('sashelp.fsp.hauto');
call send (hostcl, '_NEW_', excelobj, 0, 'Excel.Application.8');
call send (excelobj, '_SET_PROPERTY_', 'Visible','True');

/* open worksheet */
call send(excelobj, '_GET_PROPERTY_','Workbooks', wbsobj);
call send(wbsobj, '_DO_','Open', 'sales.xls');
call send(excelobj, '_GET_PROPERTY_','ActiveSheet', wsobj);

/* create a chart object */
call send(wsobj, '_COMPUTE_','ChartObjects', chobjs );
call send ( chobjs, '_DO_','Add', 144, 13.5, 287.25, 240.75 );
call send ( chobjs, '_DO_','Select' );
call send(excelobj, '_GET_PROPERTY_','ActiveChart', chartobj );

/* get the range of cells to chart */
call send(wsobj, '_GET_PROPERTY_','Range', 'C5', 'D8', rangeobj );
call send(rangeobj, '_GET_REFERENCE_ID_ ', rangeid);

/* chart the cells */
call send (chartobj, '_DO_','ChartWizard', rangeid, -4098, 6, 1, 0, 0, 1, 'Automation Demo!', 'Column', 'Value', 'Row' );
Visual Basic Code

'Launch Excel and make it visible
Private Sub LAUNCHXL_Click()
    set excelobj = CreateObject ("Excel.Application.8")
    excelobj.visible = TRUE
End Sub

'Create a new worksheet
Dim wbsobj, wobj As Object
set wbsobj = excelobj.Workbooks
wbsobj.Add
set wsobj = excelobj.ActiveSheet

'Set the value of a cell
wsobj.Cells(row+1, col).Value = var

'Close Excel
Private Sub EXITXL_Click()
    excelobj.ActiveWorkbook.Close("FALSE")
    excelobj.Quit
End Sub

OLE Automation with SCL

LAUNCHXL:
hostcl = loadclass('sashelp.fsp.hauto');
call send(hostcl, '_NEW_', excelobj, 0, 'Excel.Application.8');
call send(excelobj, '_SET_PROPERTY_', 'Visible', TRUE);

Create a new worksheet
call send(excelobj, '_GET_PROPERTY_', 'Workbooks', wbsobj);
call send(wbsobj, '_DO_', 'Add');
call send(excelobj, '_GET_PROPERTY_', 'ActiveSheet', wsobj);

Set the value of a cell
r = row+1
call send(wsobj, '_COMPUTE_', 'Cells', r, col, retcell);
call send(retcell, '_SET_PROPERTY_', 'Value', var);

Close Excel
QUITXL:
call send(excelobj, '_GET_PROPERTY_', 'ActiveWorkbook', awbobj);
call send(awbobj, '_DO_', 'Close', 'FALSE');
call send(excelobj, '_DO_', 'Quit');
call send(excelobj, '_TERM_');
return;